2005

High throughput Routing in Hybrid Cellular and Ad Hoc Networks

Ioannis Ioannidis

Bogdan Carbunar

Cristina Nita-Rotaru
Purdue University, crisn@cs.purdue.edu

Report Number:
05-006

HIGH THROUGHPUT ROUTING IN HYBRID CELLUOLAR AND AD HOC NETWORKS

Ioannis Ioannidis
Bogdan Carbunar
Cristina Nita-Rotaru

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #05-006
March 2005

High Throughput Routing in Hybrid Cellular and Ad Hoc Networks

Ioannis Ioannidis, Bogdan Carbunar, Cristina Nita-Rotaru
Purdue University
West Lafayette, IN, 47907
{ioannis,carbunar,crisn}@cs.purdue.edu

Abstract—**Hybrid networks are a promising architecture that builds ad hoc, wireless networks around the existing cellular telephony infrastructure. In this paper we present DST, a routing protocol for hybrid networks that is scalable with the network size and achieves high throughput by taking advantage of multiple channels. DST maintains a close to optimal spanning tree of the network by using distributed topology trees. DST is fully dynamic and generates only $O(\log n)$ messages per update operation. We show experimentally that DST scales well with the network size, making it ideal for the metropolitan environment hybrid networks are expected to operate in.**

I. Introduction

The past decade has witnessed rapid developments in wireless communications, from wireless cellular telephony to wireless LANs and PANs. Wireless network cards have become affordable and wireless connections have become fast enough to attract users of traditional wired communication. Current 3G implementations, e.g. of W-CDMA, provide downlink rates of up to 380Kbps, promising in the near future 2.0Mbps (2.4Mbps for cdma2000 1xEV-DO). However, the achievable rate drops significantly as the client moves away from the base station, due to path loss via distance attenuation. Furthermore, the transmission rate can be extremely erratic, making the network unreliable. While Wi-Fi hotspots are already being used to complement the coverage of cellular networks, an architecture consisting of dual, cellular and Wi-Fi equipped devices, simultaneously operating in cellular and ad hoc mode, has been proposed in [1] to improve the downlink rates of cellular clients. The model replaces direct cellular connections with freshly established paths of relayers whose cellular rates improve upon the rates of the cellular clients. Since wireless LANs offer high throughput (IEEE 802.11b [2] offers up to 11Mbps), albeit in a range of less than 200m, using a web of multihop paths can considerably increase the throughput from the base station to the devices in its cell without requiring modifications in the infrastructure. In addition, Wi-Fi standards offer multiple, non-overlapping channels.

An example of a hybrid network is shown in Figure 1. Device `A` is within the range of the base station, but the expected downlink rate is very poor, as it lies near the edge of the covered area. However, due to the higher downlink rate of `C`, the throughput of `A` can be improved by using `B` and `C` as traffic relayers. The advantage of using a dual interface is that a multihop path from a host to the base station can have a better downlink rate than the direct connection of the same host to the base station. Also, the presence of a permanent link offers possibilities for efficient routing that are not available to ad hoc networks. The cellular interface has a low capacity, but if used intelligently, it can reduce the complexity of routing.

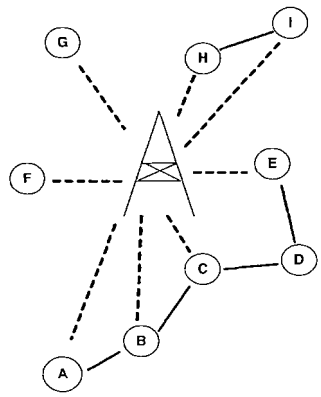

Fig. 1. Example of a multihop cellular network. The grey area represents the coverage cell of the base station. Dashed lines represent cellular links, and full lines represent Wi-Fi wireless links.

A simple solution to the problem of multihop path discovery in a hybrid network is UCAN [1]. An initiator discovers a path to the base station with a breadth-first search of the network. The protocols presented in [1] have several disadvantages: 1) flooding the network every time a path is needed can cause severe congestion; 2) when multiple hosts try to find a path to the base station, hosts that have a good downlink rate will be congested as they will be on many paths; 3) the effects of interference are ignored.

In this paper we propose DST, a routing algorithm that addresses the above problems. DST is based on a spanning tree that lazily converges to a maximum spanning tree [1]. Structuring the routing information as a spanning tree allows DST to generate only $O(\log n)$ traffic for each routing request, instead of $O(n)$ when flooding is used (n is the number of nodes in the network). This bound is achieved by using *topology trees* [3], an example of link-cut trees [4]. DST has two layers: one operates on the span-

[1] A maximum spanning tree provides the optimal routing for the next flow from a host to the base station.

ning tree, by issuing queries and update requests, the other implements these operations on the topology tree. Finally, DST exploits multiple channels to avoid interference and obtain increased throughput. Our simulations show that the achieved throughput is consistently over 80% of the optimal.

Roadmap: Section II provides details about the network model assumed in this work. Section III presents an overview of the DST protocol and the interface of its two layers. Section IV describes the operations composing the higher layer, and Section V presents the distributed implementation of the topology tree. Section VI compares the performance of our protocol with the optimal achievable throughput. Section VII places our contribution in the perspective of related work, and Section VIII presents our conclusions.

II. Network Model

We assume a wireless ad hoc network of n hosts, all situated inside the coverage area of a single cellular base station. Each mobile host is equipped with a dual cellular and Wi-Fi network card. We assume that the cellular base station can support simultaneous transmissions to/from all the hosts in its coverage area. Since mobile hosts are equipped with a single Wi-Fi transceiver, we assume that at any given time, a mobile host can communicate with at most one other mobile host. Moreover, a host cannot receive and transmit simultaneously. However, due to the dual network card, a host can support simultaneous cellular and ad hoc communications. We assume that a mobile host cannot directly adjust its Wi-Fi transmission power, but it can adjust its transmission data rate. For example, the 802.11b standard provides four transmission data rates, of 11Mbps, 5.5Mbps, 2Mbps and 1Mbps.

We model the existing network as a graph, where the mobile hosts and the base station are nodes and links denote wireless connectivity. Thus, there exists a link between the base station and each host. We consider only undirected links, since this is also an assumption of the underlying wireless MAC protocol. Each link e has a constant weight w(e) equal to its data capacity.

Similar to the work in [5] we model the interference generated by a link transmission as the set of hosts situated in the transmission range of the endpoints of the link.

To improve throughput, our protocol uses the multi-channel capabilities of wireless standards. For example, 802.11b [2] supports 3 non-overlapping channels and 802.11a [6] supports 12 non-overlapping channels. We assume that the cellular channels do not overlap with any of the Wi-Fi ones.

III. Overview of the Protocol

A. Interference and Aggregate Throughput

Consider a hybrid network where no host needs a multihop path to the base station (see Figure 2(a)). All links are available to their full capacity. If a host A needs the best available path to the base station BS and there is an optimal path discovery protocol, A can find this path and establish a flow to BS. Let this path be A, B, C, D and the capacity of the links be 11Mbps for (B, A), 5.5Mbps for (C, B), 11Mbps for (D, C) and 2.1Mbps for (BS, D). As the transmission between BS and D does not interfere with the ad hoc transmissions, the aggregate throughput of the path is the minimum between the capacity of (BS, D) and the aggregate throughput of the ad hoc path between D and A. Moreover, since each host is equipped with a single transceiver, the transmissions on (B, A) and (C, B) and also the transmissions on (C, B) and (D, C) cannot proceed simultaneously. Note also that the transmission between D and C interferes at C with the transmission between B and A. Thus, the aggregate throughput of the ad hoc path between D and A is only a fraction of the capacity of the bottleneck link, (C, B).

In the example in Figure 2 (b), the capacity of the path is 1.8Mbps. This means that A can receive data on this path at a rate decided by the capacity of the (C, B) link. This leaves links (BS, D), (D, C) and (B, A) with a residual capacity. We take a conservative approach and block any transmissions on these links for the duration of the flow introduced by A. In addition, transmissions on links of hosts adjacent to the flow path also interfere with the flow. For example, a transmission on link (G, F) interferes at F with the transmission on link (B, A) and a transmission on (F, E) interferes at B with a transmission on (C, B). We conservatively model the interference introduced on links of hosts adjacent to the flow path by blocking transmissions on them for the duration of the flow. As a result, after each time a flow is added or removed, we can deduce the state of the residual network from the physical state of the links and the sequence of flow additions and deletions.

B. Multiple Channels

Our approach of modeling the residual capacity of links due to the addition of flows and the interference they introduce is conservative and may reduce the network throughput. To overcome this problem, we take advantage of the multi-channel capabilities of the 802.11b and 802.11a wireless standards. That is, interfering transmissions can be scheduled to occur simultaneously as long as they use non-overlapping frequencies. While 802.11b offers 3 non-overlapping frequencies, 802.11a provides 12.

Selecting the transmission channel for a new flow can be done by a simple traversal of the path. For each traversed link, DST reserves the first locally available channel. Then, it contacts all the hosts whose potential transmissions interfere with the link (hosts adjacent to the link's endpoints) and reserve the chosen channel. If a link of an interfering host is left without any available channels, its residual capacity becomes 0. This process can be performed using a dedicated control channel on which all idle hosts listen. Moreover, interfering hosts are notified of the reserved channel also *on* the reserved channel, in order to discover existing, potentially interfering, communications taking place on the same channel. Figure 2(c) shows an example of this approach, where the ad hoc links supporting

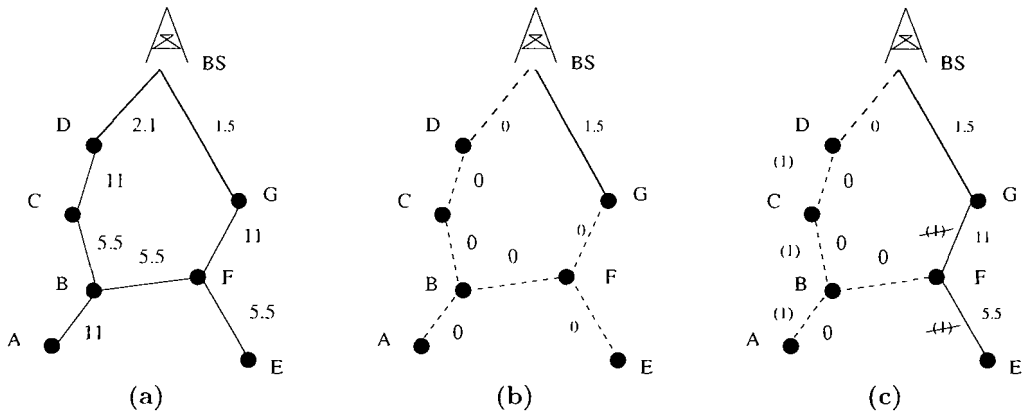

Fig. 2. (a) Example of hybrid network, where labels on the right-hand side of links represent link residual capacities. (b) Residual network of (a) after A adds a flow on links (BS, D), (D, C), (C, B) and (B, A). Due to interference, not only links adjacent to this path but also links of hosts adjacent to this path are blocked. (c) Same scenario as in (b), only using the multi-channel capability of Wi-Fi standards. Labels on the left-hand side of links represent channel assignments. Links of hosts adjacent to the flow path, i.e. (G, F) retain their capacity, but cannot use the channel chosen by A's flow due to interference.

the newly added flow of A reserve channel 1 and subsequently, channel 1 becomes unavailable for transmissions on links interfering with the flow. However, the available capacities of the interfering links are left unaltered.

C. Spanning Trees

A routing protocol that maintains the optimal path for each host in the residual network can discover multihop paths by only keeping a parent pointer for each host. In our example, the parent of A is B, the parent of B is C, the parent of C is D and the parent of D is the base station. After A adds a flow, the parent information might have to change to reflect the decrease in the capacity of the path links. At all times, the routing information constitutes a spanning tree rooted at the base station. We can prove that the maximum[2] spanning tree of a residual network provides the optimal routing information. This is a direct implication of the following well-known property of maximum spanning trees [7].

Cycle property. *For any cycle C in G, the lightest edge in C does not appear in the maximum spanning tree.*

In other words, we can construct a maximum spanning tree by deleting the worst link of every simple cycle in the network. As a result, the path from each host to the root in the spanning tree has the maximum minimum link possible. Since the aggregate throughput of a path is a fraction of the capacity of the bottleneck link of the path, this is guaranteed to maximize the capacity of the entire path.

Given a maximum spanning tree, a host can schedule the next flow by sending a forward request to its parent, which in turn will forward the request to its parent, until the base station is reached. The problem is that after adding (or deleting) a flow, the entire network may need to be contacted to derive the new maximum spanning tree, which is asymptotically not better than flooding each time a flow needs to be scheduled.

To solve the scalability issues, each time there is a change in the network, we can lazily converge to the maximum spanning tree, instead of trying to keep up with the changes. Our distributed spanning tree (DST) protocol does not change the routing information to correspond to the maximum spanning tree each time a flow is added or removed. However, each time a host requests a path, its parent pointer is set to the neighbor that has the optimal path to the base station, according to the existing information. The changes are confined in the neighborhood of a host and while queries about the state of the paths of the neighbors have to contact hosts outside the neighborhood, they can be completed much faster, as we will show. We note that if the network becomes static, the routing information will eventually converge to the optimal, even with this localized updating policy.

The advantage of this approach is that efficiently maintaining a spanning tree is possible even for large networks, as it requires only $O(\log n)$ time and messages for each operation. In the worst case, it can be arbitrarily far from the optimal, but our experiments indicate that, on average, the throughput achieved is not far from the optimal. For large and active networks, where nodes request flows frequently and links are close to capacity, DST performs extremely well.

D. Maintaining a Dynamic Tree

As we have mentioned, the entire routing protocol can be split into two layers communicating through a well-defined interface. The top layer is responsible for the maintenance of the distributed spanning tree. This layer issues a string of operations on the spanning tree. These operations are from the following set:

- Link(v, u, w) Merge the tree rooted at node v with the tree of node u by making u the parent of v. The weight

[2] We use maximum spanning trees because we want to maximize the capacity of a path. If we define a cost metric, we should instead use minimum spanning trees.

of the new link is w.

- Cut(v) Split a tree into two by removing the link of node v to its parent.
- Mincost(v) Return the minimum weight cost edge on the path from node v to the root of the tree it belongs to. This operation is called for every neighbor of a node in discoverParent (see Section IV).
- Root(v) Find the root of the tree v belongs to. This operation is useful in avoiding cycles when the Mincost (Section IV) operation is called by u for a child v in the spanning tree.
- Update(v,w) Add w to all edges on the path from v to the root of its tree. It is called by addFlow (Section IV) to reduce or increase the capacity of all the links of a multihop path.
- Cost(v) Return the cost of the edge from v to its parent.

The second layer is responsible for efficiently completing these requests. We have chosen to implement this layer with a link-cut tree. Link-cut trees are structures that can complete the above described set of operations in $O(\log n)$ time, where n is the number of hosts. This scalability property is important as it translates in $O(\log n)$ messages when implemented distributively. Furthermore, when a new host enters the network and has to query its neighbors on the capacity of their paths, the parallel time complexity is $O(\log n)$. For implementation purposes we have chosen to use topology trees for this layer. A topology tree is a relatively simple link-cut tree and it has a natural distributed implementation. We discuss topology trees and how the interface to this layer is implemented in more detail in Section V. We note that in principle any dynamic tree can be used for this layer, as long as it does not modify the structure of the spanning tree. The root of the tree is fixed to be the base station and the links are oriented towards it. Balancing operations may have to change the root and the orientation of the links and cannot be used.

IV. Description of DST

We view the hybrid cellular and ad hoc network as an undirected graph G(V,E). V is the set of nodes representing the base station (BS) and all the n hosts that it serves, and E is the set of edges. Each host h has exactly one parent p among all its neighboring hosts. Each edge $e(h_i, h_j) \in E$ has a capacity, $w(h_i, h_j)$, for example, the bandwidth available for communication between the two endpoints. Given a host h, a path $P = h_1, \ldots h_m$, where $h_1 = h$, $h_m = BS$, and host h_{i+1} is the parent of host h_i, $i = 1..m-1$, is called a *downlink path* for h. The *downlink rate of* h *through path* P is equal to one third of the minimum capacity of any link $e(h_i, h_{i+1})$, for $i = 1..m-1$ (see Section III). The optimal downlink throughput that can be achieved by h is therefore the maximum downlink rate among all paths P from h to BS. In this section we present our distributed algorithm for dynamically maintaining an approximation of the maximum spanning tree.

```
1. object implementation Host:
2.   parent: Host;
3.   minCost: double;
4.   inQ: queue; #the queue of incoming packets
5.   operation probeParent(){
6.     send(newPacket(BEACON, self), parent);
7.     startTime:=GetTime();
8.     guard inQ.first.type=HELLO do
9.       return;
10.    od
11.    guard GetTime() > startTime+T_probe do
12.      cut();
13.      discoverParent();
14.      return;
15.    od
```

Fig. 3. The definition of a host object, and operation **probeParent**.

```
16.  operation discoverParent(){
17.    bCast(new Packet(CREQ,self));
18.    startTime=GetTime();
19.    guard inQ.first.type=MCOST do
20.      m:=min(inQ.first.cost,cost(self,inQ.first.id))
21.      if m > minCost
22.        minCost:=m;
23.        parent=inQ.first.id;
24.      fi
25.    od
26.    guard GetTime() > startTime+T_disc do
27.      link(self, parent, cost(self, parent));
28.      return;
29.    od
```

Fig. 4. Operation **discoverParent**

A. Maintenance of the Spanning Tree

We describe the DST protocol using syntax inspired by the Orca [8] language. Each host h maintains information about its parent host in the spanning tree and the minimum available bandwidth, minCost, of an edge on the downlink path of h, see Figure 3. inQ represents h's queue of incoming packets, and inQ.first represents the first packet in inQ. Operation probeParent beacons h's parent (line 6). If h does not receive a reply in the alloted timeout interval T_{probe}, (lines 11-15), it first updates the topology tree by cutting the link to its parent, and then calls operation discoverParent, shown in Figure 4.

Operation discoverParent can be called by the probeParent operation, but it is also invoked periodically by each host, in order to maintain the consistency of the topology tree. This interval is called *refresh rate*. The goal of probeParent is to decide locally which neighbor of host h has the largest minCost link on its downlink path. In order to achieve this, host h broadcasts to its neighbors a cost request message, CREQ, (line 17), containing its iden-

```
30. operation addFlow(){
31.   packet:=new Packet(ADDF, self, minCost);
32.   send(packet,parent);
33.   guard inQ.first.type=ACK do
34.     update(-minCost);
35.   od
36.   guard inQ.first.type=NACK do
37.     cut();
38.     discoverParent();
39.     addFlow();
40.   od
```

Fig. 5. Operation **addFlow**

tity, and then waits for all its neighbors to answer before the timeout T_{disc} (lines 19-29).

When a host receives a CREQ message, it replies with a MCOST packet containing its minCost value. When h receives such a packet from a neighbor n (line 19), it first computes the minimum cost link on the downlink path, going through n (line 20), as the minimum between the value returned by n and the available bandwidth on the link from h to n. When the timeout T_{disc} expires, the host updates the topology tree by adding a link to its newly elected parent (lines 26-29). We note that if host h has no neighboring hosts, the base station will become its parent, maintaining the connectivity.

Whenever a host needs to download data from the base station, it calls operation addFlow, shown in Figure 5. A packet of type ADDF is created (line 31), containing the host's identifier and the host's minCost value and is sent to the host's parent (line 32), which in turn sends it to its parent, until it reaches the base station. When this happens, the base station sends an ACK packet to h.

The ADDF packet has the purpose of notifying all the intermediate relayers on the downlink path of h about their participation in the protocol. Each such relayer compares the available bandwidth on the link to its parent, with the minCost value included in the ADDF packet. If there is not enough bandwidth to accommodate the flow, the relayer contacts the base station through the cellular link, with a NACK packet containing all the information of the ADDF packet. The base station in turn contacts h with a simple NACK packet. However, if the relayer has enough capacity on the link to its parent to support the flow, it contacts all its neighbors in order to notify them of the occurring interference. Each contacted neighbor blocks its future transmissions for the duration of the flow. If the protocol takes advantage of the multi-channel capabilities of the 802.11 family of protocols (see Section III), the relayer only chooses a locally available channel for transmission and notifies its neighbors to make it unavailable for their transmissions.

After sending the ADDF packet, host h blocks waiting to receive an ACK (line 33) or NACK packet (line 36). In the former case, h updates the topology tree by removing minCost units from the available bandwidth of each link on the downlink path of h. In the latter case, h first cuts the link to its current parent in the spanning tree. Then it calls discoverParent to discover an alternate parent and the new minCost value offered by that parent. Finally, h calls again addFlow, in order to add the flow on the downlink path offered by the newly discovered parent, of rate equal to the new minCost.

The above protocol is not complete, as it is possible to create cycles. A solution is to merge Mincost and Root operations, so that not only the capacity of the path is returned, but also the root of the subtree the host belongs to. If h choosing h' as its parent creates a cycle, then MinCost for h' will return h as the root of the subtree of h'. As we have discussed, merging Root and Mincost does not increase the cost of Mincost for topology trees.

B. Refresh Rate

In dynamic networks, routes become stale quickly. A parent pointer indicating the best available path should be reevaluated at constant intervals to adapt to topology changes. The exact refresh rate depends on how dynamic the hybrid network is and how much traffic per operation we want to allow. There are two possible strategies on reassessing the parent pointer. The first is more aggressive, but generates more traffic. A node can cut its parent every k seconds and probe all its neighbors. The traffic generated is $O(d \log n)$, where d is the number of neighbors, but the parent pointer is as close to the optimal as possible with the available information. The second, less expensive, strategy is to query the parent every k seconds and cut it only if the rate falls below a threshold.

To keep the number of messages per time unit at a scalable level, we have to modulate k with the size of the network. As the network becomes more dense, the refresh rate should drop. If $k = \Theta(d)$, the traffic generated every k seconds is $O(k \log n)$. The exact constants depend on the specifics of the network, but the conclusion is that for dense networks, refreshing should be done more sparingly. This appears to be contrary to the scalability of DST, as the spanning tree maintained should be farther from the optimal. However, dense hybrid networks are more robust, and even if a link disappears, there is a high probability that an alternate equivalent link will be present. A parent pointer can be used with relative confidence for the short time until the next reassessment. In Section VI, we use an alternate approach, scaling k with $\log n$, which generates $O(d)$ messages per time unit for each host. Even at this rate, the scalability of the network is not affected, as probing one's neighbors with a heartbeat broadcast, an operation performed by almost all wireless interfaces, generates d replies. Experiments confirm that dropping the refresh rate according to $\log n$ does not affect the performance of DST.

V. Topology Trees

The characteristic of topology and, in general, link-cut trees is that they can maintain dynamic trees of n nodes

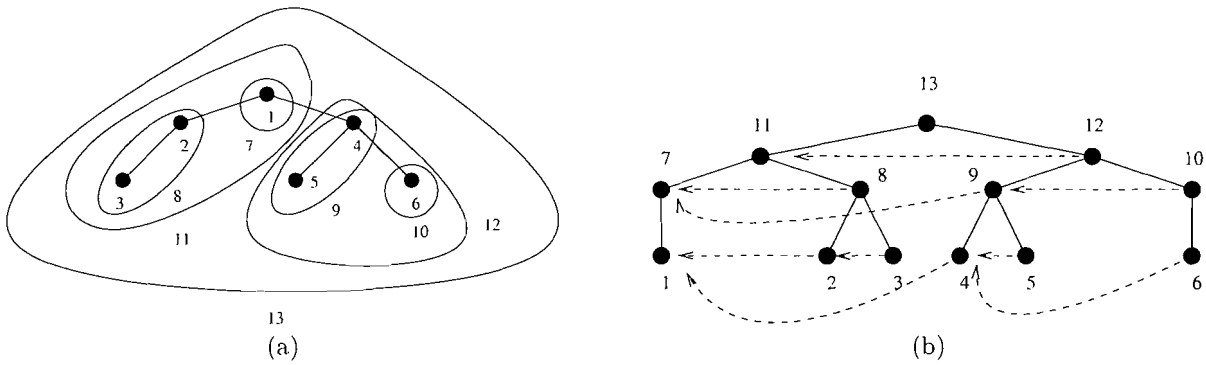

Fig. 6. (a) Example of a restricted partition and (b) the resulting topology tree.

with $O(\log n)$ operations per tree update. Also, operations that require an aggregate of all the nodes on a path from the root to a leaf, like `Mincost` or `Update`, can be completed in $O(\log n)$ time. In fact, all operations of the interface to the second layer have $O(\log n)$ time and message complexity.

The reason for choosing topology trees is that they are conceptually simpler than the more common splay trees. They are also naturally distributable structures and asymptotically they are optimal. Experimental results from [3] indicate that although they are less efficient than splay trees, the difference does not offset their implementation advantages. We will give a brief overview of topology trees in this section. For a detailed presentation and an example of topology trees supporting a complicated minimum spanning tree algorithm see [9].

A. Overview

Topology trees are derived from a restricted partition of a tree. For an example of a restricted partition see Figure 6(a). To avoid confusion, we will refer to nodes of the topology tree as clusters. The leaves of the topology tree are clusters of single spanning tree nodes. A cluster of a higher level is made up of one or two clusters of a lower level. The rules by which clusters are paired are described in [3]. The intuition is that for every pair of clusters that combines for a cluster of a higher level, another cluster is made up of a single lower level cluster to act as a buffer when there is an update in the structure of the topology tree. These buffers are clusters that have two children. In Figure 6(a), cluster 7 consists of only cluster 1 for this reason. It can be shown that the height of a topology tree is $O(\log n)$ [3].

The resulting topology tree is shown in Figure 6(b). The solid edges indicate the relationship between clusters of consecutive levels. The dashed arrows represent the structure of the tree formed by clusters of the same level. Each such tree is called the induced tree of the specific level. The lowest level induced tree is the actual spanning tree. Observe that a cluster with two induced children will have only one child in the topology tree.

Besides the adjacency information, each cluster stores three more fields, `Δcost`, `nodemin` and `minvert`. If we want to calculate the weight of an edge from node v to its parent, we need to traverse the topology tree from the leaf cluster corresponding to v up to the root of the topology tree and sum the `Δcost` fields of the accessed clusters. The `nodemin` and `minvert` fields of a cluster c store information about the minimum cost edge in the spanning subtree induced by the leaves of the subtree rooted at c. The rules by which these fields are calculated and how the adjacency information should change after an update of the spanning tree can be found in [3] and in more detail in [9]. We note that each of the $O(\log n)$ steps is a local operation that only needs information from the parent and sibling cluster to complete. This is important not only because it leads to $O(\log n)$ total time for each of the operations of the topology tree, but also because it facilitates the distributed implementation of topology trees.

B. Distributed Topology Trees

A naive implementation of topology trees can assign the responsibility for maintenance to the base station which acts as the root of the spanning tree. The problem is that the base station can become the bottleneck of a large network, where update requests are frequent. We can implement topology trees in a simple and efficient way by assigning each cluster to a node in the network and using the base station only as a relayer of messages. The entire protocol is quite long, but not overly complicated. For lack of space, we will present only the logic of a distributed implementation.

For a topology tree operation, like `Mincost`, for each level of the topology tree a single cluster needs to be contacted. We assign a head for each cluster. This is the node that will be responsible for all the operations regarding this cluster. Node v is responsible for the leaf cluster corresponding to v. There is only a limited amount of information required for maintaining a cluster, namely five pointers to the heads of the adjacent clusters, a pointer to a node for the `minvert` fields and two reals for `Δcost` and `nodemin` fields. There are $O(n)$ clusters, which we can map so that each node is the head of a constant number of clusters. Load balancing is a concern when we map the

clusters. The higher the cluster is placed in the topology tree, the more work will be directed towards its head. A solution could be to periodically randomly reassign clusters. The only restriction for a mapping is that a leaf cluster is assigned to the corresponding node, so that the initial cluster for an operation can be located quickly.

Each cluster needs only information from its "neighborhood" to complete an operation. For example, during `Mincost`, the cluster responsible for level `i` needs information for its sibling in the topology tree. To receive this information, it needs to contact its parent, which will contact the sibling, which will send its information. These messages are short, as they communicate the status of a cluster (adjacency information, fields). They can be carried by the low bandwidth channel between a node and the base station. The base station needs only to relay these messages to their recipient cluster heads. In total, each time two clusters need to contact each other, at most four short messages are generated. To complete each operation, $O(\log n)$ messages are sent.

C. Synchronization

The distributed nature of the topology trees used in our scheme raises issues of realizability and consistency. We can solve such problems with the use of locks. We want the locks to be applied to the finest level possible. A simple but inefficient solution would be to lock the entire structure each time a node initiates an operation that involves the topology tree. This can be achieved by having the base station queuing all operations until the current DST operation is completed. However, the tree structure offers itself for more efficient locking schemes.

Suppose node 4 and node 6 in the tree of Figure 6(a) both want to execute an operation on the topology tree. For example, node 4 could do an `Update` while node 6 initiates a `Mincost`. This would correspond to node 4 adding a flow to the base station and node 6 inquiring about the capacity of its path to the base station, which goes through node 4. By examining the topology tree of Figure 6(b), we see that both operations can proceed independently up to cluster 12[3]. If the `Update` operation reaches cluster 12 first, `Mincost` will have to wait until `Update` finishes with cluster 12. Node 4 can even start another operation before completing its `Update`.

D. Fault Tolerance

By assigning each cluster to a node we solve the bottleneck problem, but we have created the need for a mechanism that can recover the cluster information when the responsible node goes down. The easiest recovery mechanism is to replicate the clusters in the base station. If a host does not respond to a request for a lock, a lock is held for a long time or a child cannot reach its parent, the base station can be contacted. The duplicate of the cluster can be used to either have the base station complete the operation or assign a new cluster head, which will complete the operation of cutting the missing host from the network. As we have discussed, each cluster stores only limited information and duplicating it to the base station, which already has information regarding each host logged in the network, is well within the capabilities of a cellular base station.

[3] Actually, `Update` will need to lock cluster 10 when it reaches cluster 9. Nevertheless, the point we want to make is that locks need to be acquired only for a small neighborhood of a cluster.

VI. Simulation Results

In this section we present an experimental analysis of the throughput performance of DST with regard to the optimum throughput achievable when a centralized knowledge Bellman-Ford algorithm is executed and to the basic cellular rate available to mobile hosts. We have performed extensive simulations involving mobile hosts and multiple flows, initiated concurrently by multiple hosts.

A. Simulation environment

In our experiments we model the ad hoc network using the unit disk graph model and the Agere Short Antenna PC Card Extended specification. We use the ARF [10] mechanism to establish the transmission rate of the communication channel between two mobile hosts. We use only the top two transmission rates, of 11Mbps for distances under 160m and of 5.5Mbps for distances under 270m. The hosts are initially deployed randomly in a square of area $2830 \times 2830\text{m}^2$ and we use a modified random waypoint model [11] to simulate their movements. We have overestimated the effects of ad hoc link interference, by blocking any transmission involving hosts adjacent to the link.

We use the UCAN [1] approach to model the dependency between the cellular link rates of hosts and their distance from the cellular base station. The base station is positioned at the center of the $2830 \times 2830\text{m}^2$ deployment square and its cellular transmission range is 1920m. According to this model, each host inside the square is covered by the cellular transmission range of the base station.

We model the optimal throughput to be the one achieved by running the Bellman-Ford algorithm. Instead of computing the shortest path, we compute the maximum throughput path between the base station and all the mobile hosts it covers. In the case of multiple concurrent flows the optimal for n flows is computed by running Bellman-Ford on the residual network obtained after removing the bandwidth consumed and the interference introduced by the first $n-1$ flows.

We perform each experiment by choosing 5 different initial network configurations. For each such configuration the experiment is run for 100 seconds. Thus, each point on the plots is an average over 500 measurements.

B. Single flow

The following experiments assume that the only host accessing the base station is situated initially at a distance of 1280m from the base station. However, during the experiment, all the hosts, including the client host, are mobile. In the legend of the plots, "DST single ch" is used to denote DST when using the standard, single channel mode of

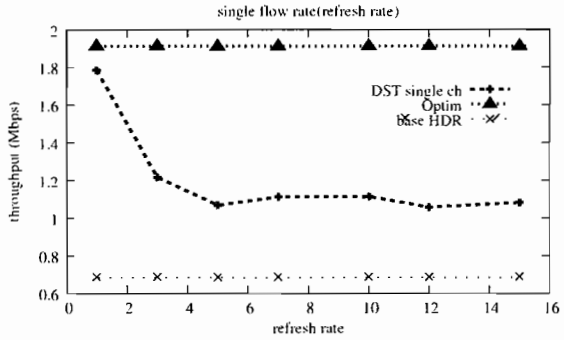

Fig. 7. Evolution of DST for a single flow when the refresh rate, k, increases from 1 to 15s. The total number of hosts is 300 and their maximum velocity is 9m/s. The lower line represents the base cellular rate and the upper line represents the optimum achieved by Bellman-Ford.

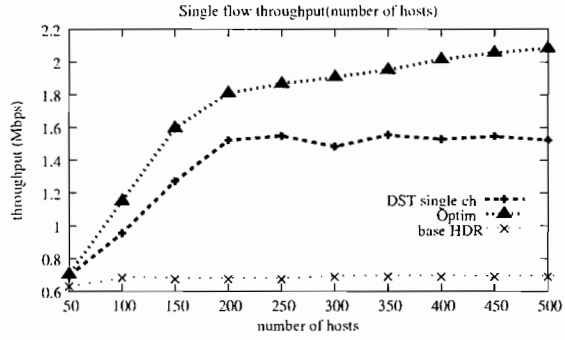

Fig. 9. The throughput of DST and the optimum achieved by Bellman-Ford, for a single flow, as a function of the number of hosts, for a constant maximum speed of 9m/s. The refresh rate of DST is $\log n$, where n is the total number of hosts, thus increasing for this experiment from 6 to 9s.

802.11b, "Optim" denotes the output of Bellman-Ford and "base HDR" represents the basic cellular rate available to the client host by default.

The first experiment measures the dependency between the throughput achieved by DST and the refresh rate of the hosts, when the total number of hosts is 300. As expected, DST performs best for the smalled refresh rate value and then its performance degrades, but it stabilizes at $k = 3$. However, as specified in Section IV, for a value of $k = \log n$, where n is the number of hosts, the algorithm generates less traffic (order of the degree of the network messages per time unit per host). In this case, $k = 9$, and the throughput improvement of DST over the base cellular rate is still significant (430kbps).

The second experiment compares the evolution of the throughput achieved by DST when the maximum velocity of the hosts increases from 3 to 30m/s, for a total of n=300 hosts served by the base station, with Bellman-Ford and the basic cellular rate of mobile hosts. For DST we perform the experiment with a refresh rate value, k, set to $\log n = 9$. Figure 8 shows that the performance of DST follows the trend of the basic cellular rate and is very close, between 75 and 85%, of the optimum throughput achieved by Bellman-Ford. Since we only experiment with a single flow in the system, there exists no interference due to other traffic in the network.

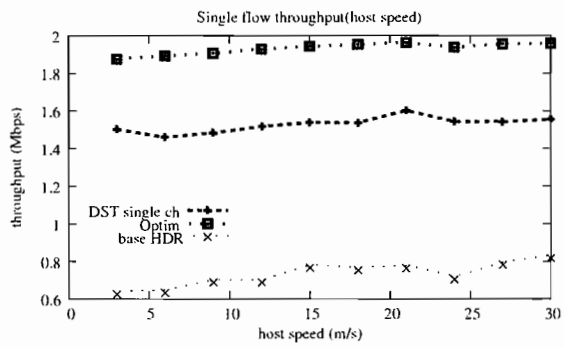

Fig. 8. Evolution of the throughput achieved for a single flow by DST and Bellman-Ford, for 300 hosts when the maximum velocity of each host increases from 3 to 30m/s. k, the refresh rate of DST, is set to $\log n$, which in this scenario is 9s.

Finally, we explore the relationship between the throughput achieved by DST, Bellman-Ford and the cellular rate available to mobile hosts and the density of hosts served by the base station. In this experiment all the hosts move at a maximum velocity of 9m/s. We measure the evolution of the throughput achieved by the client host when the total number of hosts deployed in the same square grows from 50 to 500. DST is again evaluated with a refresh rate, $k = \log n$, that for this experiment ranges between 6 and 9s. Figure 9 shows the throughput of DST compared with the optimal achievable throughput. It can be observed that DST scales very well with the number of mobile hosts, achieving between 75 and 98% of the optimum.

C. Multiple concurrent flows

We investigate the performance of DST when multiple clients support flows simultaneously. We use as baseline the cellular data rate of the clients and we present the results as percentage of the optimum throughput achieved by Bellman-Ford.

In addition to the performance of DST when a single transmission channel is used, we also experiment with multi-channel transmissions. First, we use the 3 non-overlapping channels of 802.11b, denoted on the plots as "DST 3 channels". Secondly, we switch to the 802.11a specification, providing 12 orthogonal channels and transmission rates of up to 54Mbps. However, for our simulations we use only the 54Mbps and 48Mbps links. We label the results of DST using 802.11a with "DST 12 channels". As before, we use "DST single ch" to denote the standard single channel operation mode using 802.11b and "base HDR" represents the cellular rate available to the client hosts.

In the first experiment we randomly deploy 300 hosts that continuously move with a maximum speed of 9m/s.

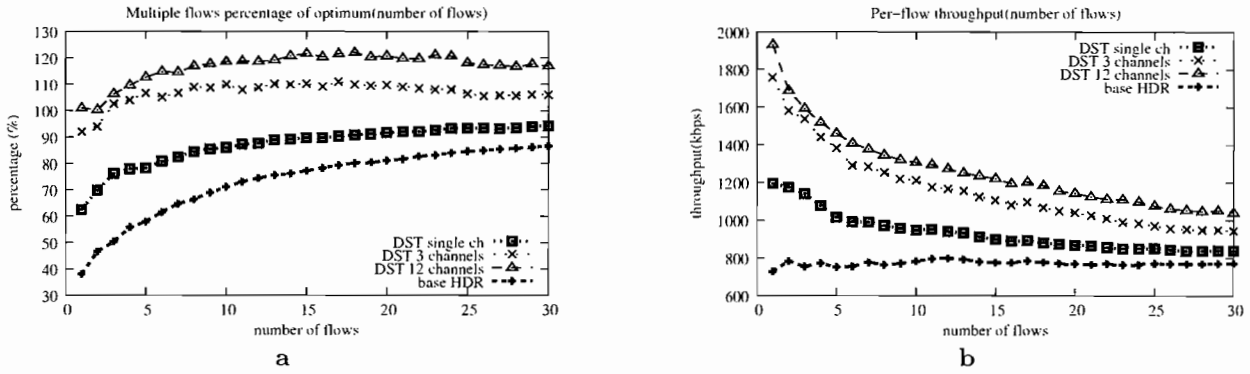

Fig. 10. (a)The throughput of DST and the basic cellular rate as percentage of the throughput achieved by Bellman-Ford when the number of flows grows from 1 to 30 for a network of 300 hosts. We show the results for DST run using 802.11b in single channel mode, 802.11b in 3-channel mode and 802.11a using all 12 channels. (b)The per-flow throughput corresponding to (a).

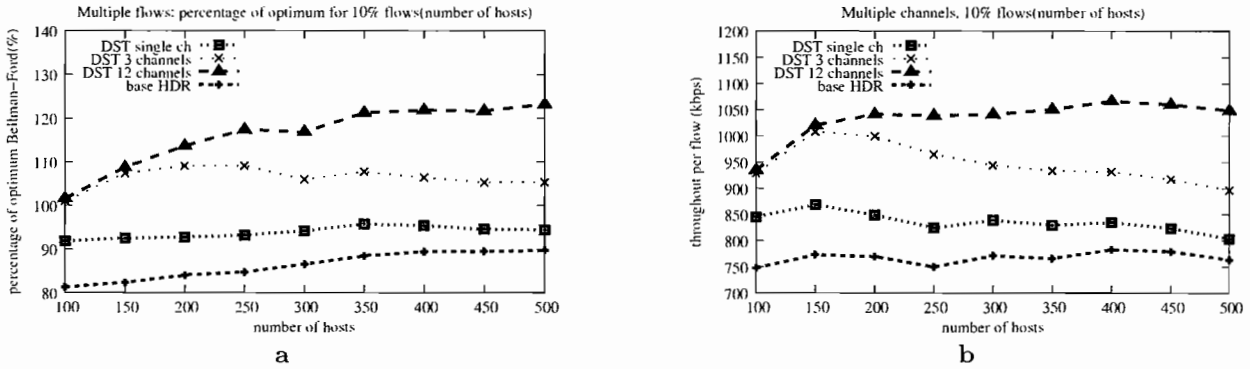

Fig. 11. (a)The throughput of DST and the basic cellular rate as percentage of the throughput achieved by Bellman-Ford for networks of 50 to 500 hosts, when 10% of the hosts concurrently hold a flow. The results for DST run in conjunction with 802.11b single channel, 802.11b 3-channel and 802.11a are shown. (b) The per-flow throughput corresponding to the results in (a).

We increase the number of simultaneously supported flows from 1 to 30. Figure 10(a) shows the performance of DST relative to the optimal total throughput, achieved when all the client hosts run the distributed Bellman-Ford algorithm to find the best downlink path. The performance of DST increases to achieve more than 90% of Bellman-Ford. In addition. Figure 10(a) also shows the performance of DST when using the multi-channel capabilities of 802.11b and 802.11a, compared to the flat cellular rate of the client hosts. Using the non-overlapping channels of 802.11b brings an increase of around 10% over the optimum achievable in the case of a single channel, while 802.11a has a 20% increase. Note that even when 30 out of the 300 hosts concurrently support a flow, by using the 3 channels of 802.11b, DST achieves a per-flow increase of 200kbps over the basic cellular rate (see Figure 10(b)).

The second simulation experiments with increasing concentrations of mobile hosts and of concurrent flows. In the same square area of $2830 \times 2830\mathrm{m}^2$, we place between 100 and 500 hosts, while also increasing the number of hosts concurrently supporting flows to be 10% of the total number of hosts. Figure 11(a) shows that DST performs very close to the Bellman-Ford, always higher than 90%. Using the 3 channels of 802.11b brings a 10% increase over the single channel variant, whereas using 802.11a achieves a throughput increase of up to 25% over the optimum Bellman-Ford. Figure 11(b) shows the results in terms of the per-flow throughput. While the basic cellular rate remains constant, as the network becomes congested, the throughput achieved by DST per flow gracefully decreases when using the single channel or the multi-channel capabilities of 802.11b. However, when using DST in conjunction with 802.11a, the throughput per-flow saturates at 1050Kbps. This is because the usage of multiple non-overlapping channels alleviates the effects of the congestion generated at the hosts situated in the vicinity of the base station, by allowing concurrent transmissions on their adjacent hosts. Using DST with the 3-channel variant of 802.11b, brings an increase of between 150 and 200kbps over the cellular throughput. When using DST in conjunction with 802.11a, the throughput increase is more substantial, between 200 and 300kbps.

VII. Related Work

The most popular model of wireless networks in the literature is that of the ad hoc architecture [12], [13], [11].

The entirely distributed nature of ad hoc networks limits their scope, as maintaining a connected network over a large area is quite difficult. There have been efforts to integrate infrastructure-based network models with ad hoc components, but most of them assume single-interface devices. In [14], GSM terminals are used to relay information to other terminals to improve coverage. In Opportunity Driven Multiple Access (ODMA) [15], transmission power is conserved by relaying traffic from a CDMA host to the base station through multiple, short hops. In [16], some channels are reserved for forwarding when the fixed channels become congested. In [17], a generic wireless network is considered, where hosts contact a mobile base station for access outside their cell, using only one interface. In [18], a hybrid network using the IEEE 802.11 [19] architecture with both DCF and PCF modes is examined, using only one wireless interface. In [20], multihop paths are used to decrease the number of base stations by increasing their coverage. The overall capacity increases only when two communicating hosts are in the same cell.

Although double-interface architectures are conceptually similar to their single-interface counterparts, they increase the overall capacity by using short-range, high-bandwidth, ephemeral channels to relay traffic and a long-range, low-bandwidth, permanent channel to complete operations like routing and data integrity confirmation or as a last resort in the absence of neighbors. The low-bandwidth channels are not necessarily cellular, but the already existing infrastructure makes them attractive options. This architecture has been examined in [1]. In [21], traffic is diverted to neighboring cells to increase throughput. The use of dedicated, stationary relays increases the cost of their solution and limits its utility. In [22], wireless nodes in a mesh network are equipped with two Wi-Fi network interface cards. Centralized channel assignment algorithms and a routing protocol, designed to increase the aggregate throughput in the presence of interference, are presented.A study of local area hybrid networks is presented in [23]. A comprehensive presentation of a rudimentary hybrid network can be found in [24].

The problem of maintaining dynamic spanning trees is well-studied. In the context of ad hoc networks we are interested in the complete dynamic model, where hosts can turn on and off arbitrarily, in addition to edge deletions and insertions. This is the most powerful model of dynamic networks. In [25], a fully dynamic minimum spanning tree is maintained in $O(n^{1/3} \log n)$ time per update. We note that topology trees can be used to maintain a minimum spanning tree in $O(\sqrt{m})$ time, where m is the number of edges. For planar graphs, maintaining a minimum spanning tree is more efficient and can be achieved with $O(\log n)$ time per update [26].

Routing with the use of a spanning tree of the network is a natural technique and some routing protocols have been proposed for ad hoc networks in [27] and [28]. Their utility, however, is severely limited when the hosts are mobile. For static, ad hoc, wireless networks spanning trees have been studied as the basis of power-aware routing protocols, especially for broadcasting purposes [29], [30].

VIII. Concluding Remarks

We have described and evaluated analytically and experimentally the DST protocol for discovering multihop paths in hybrid networks. The strength of DST is its scalability. In metropolitan areas, where a cell may need to serve hundreds of mobile hosts requesting Internet access, it is crucial that routing has a low time and message complexity and that its performance does not suffer as the network size increases. DST exhibits all these characteristics, by maintaining a spanning tree of the network that is close to optimal, but without the overhead of being exactly the optimal. By using topology trees to maintain the dynamic spanning tree, each operation can be completed in $O(\log n)$ time, generating $O(\log n)$ messages. The total throughput is constantly close or over 80% of the optimal routing for active networks. Solutions relying on flooding are unscalable regarding both complexity and performance, as our experiments indicate.

While maintaining a maximum spanning tree for the residual network may be unachievable, due to its complexity, a solution can come from simplifying the network. Maintaining the maximum spanning tree for planar graphs has a much lower complexity. Exploiting the characteristics of the physical links, it may be possible to discover planar subgraphs of the network graph, where the maximum spanning tree for the subgraph is provably close to the overall maximum spanning tree.

References

[1] H. Luo, R. Ramjee, P. Sinha, L. E. Li, and S. Lu, "Ucan: a unified cellular and ad-hoc network architecture," in *ACM MOBICOM*, pp. 353–367, 2003.

[2] *IEEE Std 802.11b-1999*. 1999. http://standards.ieee.org/.

[3] G. N. Frederickson, "A data structure for dynamically maintaining rooted trees," in *SODA*, 1993.

[4] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. Sci.*, vol. 26, pp. 362–391, 1983.

[5] M. Burkhart, P. von Rickenbach, R. Wattenhofer, and A. Zollinger, "Does topology control reduce interference?," in *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pp. 9–19, ACM Press, 2004.

[6] *IEEE Std 802.11a-1999*. 1999. http://standards.ieee.org/.

[7] R. E. Tarjan, *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, 1983.

[8] H. Bal, M. Kaashoek, , and A. Tanenbaum, "Orca: A language for parallel programming of distributed systems," *IEEE Transactions on Software Engineering*, vol. 18, pp. 190–205, March 1992.

[9] G. N. Frederickson, "Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees," *SIAM J. of Comp.*, vol. 26(2), pp. 484–538, 1997.

[10] A. Kamerman and L. Monteban, "Wavelan-ii: A high performance wireless lan for the unlicensed band," *Bell Labs Technical Journal*, 1997.

[11] D. Johnson and D. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing, volume 353, pages 153-181. Kluwer Academic Publishers*, 1996.

[12] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *ACM MOBICOM*, pp. 85–97, 1998.

[13] C. Perkins and E. Royer, "Ad-hoc on demand distance vector routing," in *IEEE WMCSA*, 1999.

[14] G. Aggelou and R. Tafazolli, "On the relaying capacity of next-generation gsm cellulat networks," February 2001.

[15] T. Rousse, I. Band, and S. McLaughlin, "Capacity and power investigation of opportunity driven multiple access (odma) networks in tdd-cdma based systems," 2002.
[16] X. Wu, S.-H. Chan, and B. Mukherjee, "Madf: A novel approach to add an ad-hoc overlay on a fixed cellular infrastructure," in *Proceedings of IEEE WCWN*, 2000.
[17] I. Akyildiz, W. Yen, and B.Yener, "A new hierarchical routing protocol for dynamic multihop wireless networks," in *IEEE INFOCOM*, 1997.
[18] H.-Y. Hsieh and R. Sivakumar, "On using the ad-hoc network model in wireless packet data networks," in *ACM MOBIHOC*, 2002.
[19] IEEE, *IEEE Std 802.11. 1999 Edition.* 1999. http://standards.ieee.org/catalog/olis/lanman.html.
[20] Y.-D. Lin and Y.-C. Hsu, "Mulithop cellular: A new architecture for wireless communications," in *Proceedings of IEEE INFOCOM*, 2000.
[21] S. De, O. Tonguz, H. Wu, and C. Qiao, "Integrated cellular and ad hoc relay (icar) systems: Pushing the performance limits of conventional wireless networks," in *HICSS-37*, 2002.
[22] A. Raniwala, K. Gopalan, and T. cker Chiueh, "Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, no. 2, pp. 50–65, 2004.
[23] S. Lee, S. Banerjee, and B. Bhattacharjee, "The case for a multihop wireless local area network," in *IEEE INFOCOM*, 2004.
[24] M. Miller, W. List, and N. Vaidya, "A hybrid network implementation to extend infrastructure reach," tech. rep., University of Illinois at Urbana Champaign, January 2003.
[25] M. Henzinger and V. King, "Maintaining minimum spanning trees in dynamic graphs," in *ICALP*, 1997.
[26] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung, "Maintenance of a minimum spanning forest in a dynamic planar graph," in *SODA*, 1990.
[27] S. Radhakrishnan, G. Racherla, C. N. Sekharan, N. S. V. Rao, and S. G. Batsell, "Protocol for dynamic ad-hoc networks using distributed spanning trees." *Wireless Networks*, vol. 9, pp. 673–686, 2003.
[28] R. Sivakumar, B. Das, and V. Bharghavan, "Spine routing in ad hoc networks." *Cluster Computing*, vol. 1, no. 2, pp. 237–248, 1998.
[29] J.-H. Chang and L. Tassiulas, "Energy conserving routing in wireless ad-hoc networks," in *IEEE INFOCOM*, pp. 22–31, 2000.
[30] P. Wan, G. Calinescu, X. Li, and O. Frieder, "Minimum-energy broadcasting in static ad hoc wireless networks," *Wireless Networks*, vol. 8, no. 6, pp. 607–617, 2002.